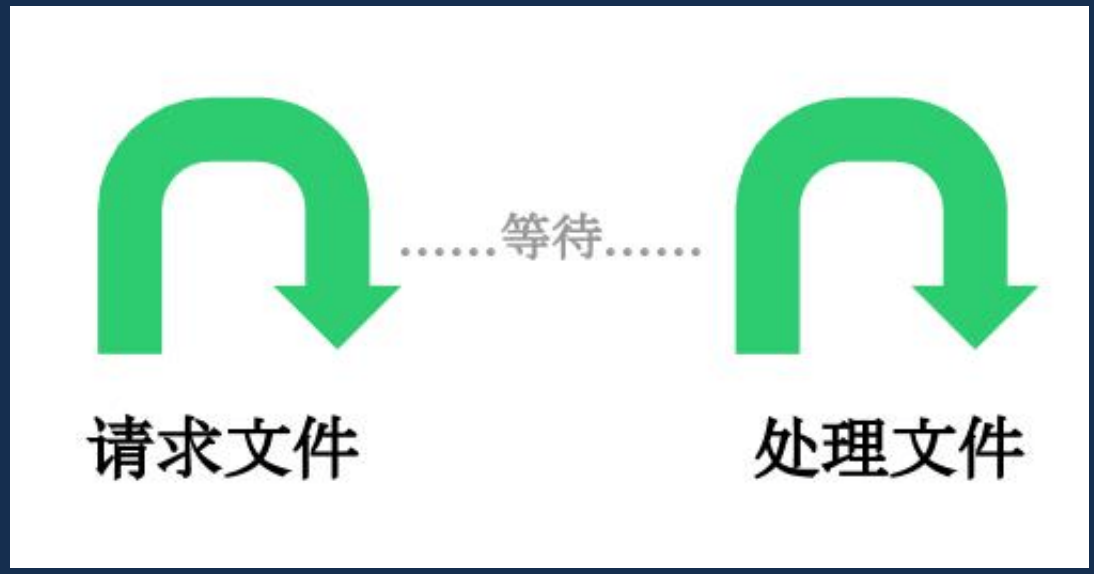
node之异步流程控制

—刘晓阳

同步和异步



node带来的好处

- 高并发
- 非阻塞I/O
- 轻量高效
- 等等...

代码异步的方式去执行

```
tf_api(query, function(err, val1){
    //do something
});
```

.....

```
tf_api(query, function(err, val1){
   //....do something
   base_api(val1, function(err, val2){
      //....do something
      tms_api(val2, function(err, val3){
         //....do something
         fin_api(val3, function(err, val4){
            //....do something
            return data_info;
         });
      });
   });
});
```

那有没有解决方法?

链式调用

```
var action = {
  a:function(){
    console.log('a');
    return this;
  },
  b:function(){
    console.log('b');
    return this;
  },
  c:function(){
    console.log('c');
    return this;
  },
  d:function(){
    console.log('d');
    return this;
  }
};

action.a().b().c().d();
```

链式调用好处

- 每一个操作都是独立的函数
- 可组装，拼就好了

但是这还不够

- 下一步获得上一步的输出结果
- 出错能捕获异常
- 在函数里处理业务流程

目前系统里在用的是Seq

```
new Seq()
    .seq(function(){
        tf_api({tid : 123}, this);
    })
    .seq(function(task){
        customer_api({cuid : task.cuid}, this);
    })
    .seq(function(customer){
        var that = this;
        fin_api(customer, function(err, result){
            if(err) that(err);
            cb(null, result);
        });
    })
    .catch(function(err){
        cb(err);
    });
....
```

并行

```
new Seq()
    .par('task', function(){
        tf_api({tid : 123}, this);
    })
    .par('customer', function(){
        customer_api({cuid : 456}, this);
    })
    .seq(function(){
        var that = this,
            task = this.vars.task,
            customer = this.vars.customer;
        driver_api({did : 789}, function(err, ret){
            if(err) return that(err);
            return cb(null, {...});
        });
    })
    .catch(function(err){
        cb(err);
    });
....
```

还有没其他的?

Promise/A+

先看下Promise的

```
//刚才的链式操作
action.a().b().c().d();

//promise
action().then(a).then(b).then(c)
```

Promise是把函数传入他的then方法

如果失败呢?

```
action().then(a).then(b).then(c).catch(function(err){
    //do something
});
```

Promise的应用

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
var Promise = require("bluebird");

UserSchema = new Schema({
  username: String,
  password: String,
  created_at: {
    type: Date,
    "default": Date.now
  }
});

var User = mongoose.model('User', UserSchema);


Promise.promisifyAll(User);
Promise.promisifyAll(User.prototype);
```

查询

```
    User.findAsync({username: username}).then(function(data) {
    ...
}).catch(function(err) {
    ...
});
```

ES6上的Generators/yield

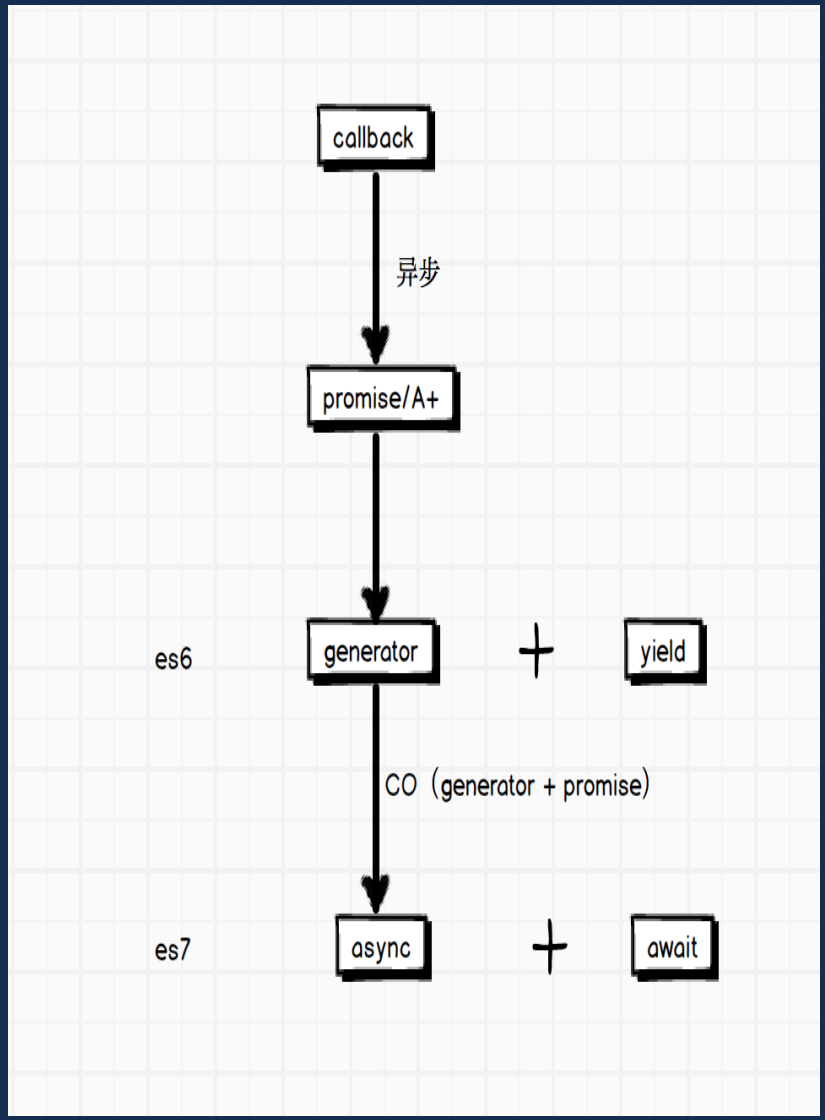
```
function* gen () {
    yield 'hello';
    yield 'world';
}

var g =  gen(); // 返回的其实是一个迭代器


console.log(g.next());    // { value: 0, done: false }
console.log(g.next());    // { value: 1, done: false }
console.log(g.next());    // { value: undefined, done: true }
```

co + Promise

co是著名的tj大神写的，是一个为Node.js和浏览器打造的基于生成器的流程控制工具，借助于Promise，你可以使用更加优雅的方式编写非阻塞代码。

```
var task = yield tr_api(query);

var customer = customer_api(task.cuid);
var drivers = driver_api(task.bids);
var citys = citys_api(task.citys);
var bid_mgr = admin_user_api(task.bid_mgr_id);
var warehouse  = warehouse_api(task.wid);

var result = yield [
        customer,
        drivers,
        citys,
        bid_mgr,
        warehouse
    ];

return result;
}).then(function(info){
   .... //do something
}).catch(function(err){
    console.log(err);

    return res.sendError('服务器错误');
```

未来， Async/await



参考资料

参考资料： Promise/A+规范 Node.js技术栈之Promise

一起来实现co 简单实现Promise/A+

Generator 函数的含义与用法